Protocol Orchestration: A Semantic Approach to Communication Stacks*

Stefan Götz, Tobias Heer, Klaus Wehrle
Distributed Systems Group
RWTH Aachen University, Germany
firstname.*lastname*@cs.rwth-aachen.de

ABSTRACT

The diversity of today's networking environments, such as wired, wireless, cell-based, or multi-hop, is matched by an equally large amount and heterogeneity of specialized protocols, e.g., overlays, Wi-Fi positioning, MANET routing, cross-layer signaling. However, communication is typically performed with a static set of protocols selected at design time based on simplified assumptions ignoring the environment's heterogeneity.

In this paper, we argue that protocols can be orchestrated as software components at run time driven purely by their functionality and the demands of the execution environment. Our end-system protocol framework ADAPT bases on extensible ontological models that semantically describe protocol and environment properties. Each connection receives a custom-tailored protocol stack that ADAPT orchestrates from the requirements derived from the application, user, and environment. With this approach, end-systems can reason about the functionality and quality of automatically composed and adapted protocol compounds while remaining open to existing and future protocols.

Categories and Subject Descriptors

C.2.1 [**Computer Systems Organization**]: Computer-Communication Networks—*Network Architecture and Design*

1. INTRODUCTION

The classic TCP/IP protocol stack is static which has raised a large number of practical and scientific challenges over recent years. In particular, the difficulties in deploying new protocols and flexibly integrating them with each other on end systems hinders their proliferation. Protocols like IPv6, HIP [5], or SCTP [12] crucially depend on standardization and inclusion in major operating systems instead of being readily available to users when in demand. Similarly, new or custom protocols like MANET protocols, wireless TCP extensions, or VPNs do not integrate naturally with the TCP/IP stack, depend on non-standard OS features, and are cumbersome for users to install and configure.

The ossification of TCP/IP in particular affects users of mobile communication and entertainment devices. Typically, network hand-overs interrupt existing connections and frequently require the user's attention. Users need to manually pick the appropriate network attachment (e.g., wired vs. wireless vs. cellular), configure it (e.g., activate a VPN), authenticate, and be constantly aware of factors like costs, privacy, and security. Thus, the user experience is far from the vision of permanent and adaptive connectivity.

Ideally, a communication subsystem should rather resemble a configurable, dynamic framework of individual protocols that, in their entirety, provide a desired functionality with the most "suitable" characteristics. The "suitability" of a communication subsystem depends on a wide variety of dynamic factors, including those well-known from Quality-of-Service (QoS) literature: application requirements (e.g., reliability, latency, security), device capabilities (e.g., CPU, memory, and energy constraints), and network characteristics (e.g., loss rate and throughput). Additional important factors arise from user preferences (security, cost, etc.), the network configuration (e.g., requiring authentication or tunneling), and the capabilities of the communication partners (such as their support for specific protocols). All of these criteria can influence how individual protocols need to be orchestrated into a functionally sound and well adapted compound. The length and incompleteness of this list obviates the main deficiency of existing approaches to structuring protocol frameworks. They describe protocols through strict design-time classifications, such as class hierarchies or languages with a fixed vocabulary, and are thus only poorly extensible. Thus, they cannot incorporate new protocols, functionalities, or requirements. However, this extensibility is crucial for escaping the ossification trap and supporting, for example, layer violations and extensions as in HIP and new cross-cutting aspects like WiFi positioning.

In this paper, we discuss our dynamic protocol framework ADAPT. It leverages a semantic description format based on ontologies that provides an abstract notion of a protocol's functionality and properties. This format is the corner stone for ADAPT to be extensible with new protocols and requirements, including novel, e.g., non-layer, network architectures. It also makes it possible to drive the orchestration

*This work is funded by the German Research Foundation DFG under the project ADAPT (WE 2935/4-1)

MobiArch'09, June 22, 2009, Kraków, Poland.

of protocols into protocol stacks by application and environment requirements. ADAPT offers a generic protocol model that is not bound to established layering conventions and consistently integrates tunneling, encapsulation, and transformation (e.g., for VPNs, overlays, encryption).

2. RELATED WORK

Existing research addresses the area of dynamic communication stacks from a number of different angles. As many early papers reflect, flexibility is classically achieved through componentization, typically motivated by re-use, reduction of complexity, customization, and performance. Language-based approaches such as COMSCRIPT [13, 6] introduce language constructs to enforce uniform component interfaces and to achieve component instantiation at compile time and run time. To avoid the intrusive dependency on a particular programming language, the x-Kernel [9, 2] settles on a small common component interface by convention and introduces fundamental abstractions for functional building blocks (i.e., protocols), messages, and sessions (i.e., protocol and connection-specific data). In [7], O'Malley and Peterson argue in particular for strictly confining knowledge about protocol functionality in the orchestration of components rather than in their implementations to increase modularity. ADAPT builds on these foundations for genericness, flexibility, and customizability without necessarily promoting protocol decomposition, novel user/kernel interfaces, etc.

Observing how protocol layering is violated in existing systems (e.g., with tunneling and overlay networks) and how it limits protocol flexibility, Braden and colleagues take protocol decomposition one step further. Their Role Based Architecture [1] abandons layering and resorts to packet processing in loose ordering based purely on functional necessities at a granularity of functional components smaller than full protocols like TCP. In ADAPT, we follow the idea of communication systems being driven by functionality rather than coarse-grained conventions such as layers.

The composition of a protocol stack needs to follow guidelines like protocol dependencies or functional requirements (for example to perform compression before encryption). Custom languages help to explicitly separate this information from the implementation, as shown by F-CSS [16] and DaCapo [10]. However, the separation remains incomplete with such description formats since they expose implementation aspects. Also, they rigidly encode the protocol-related knowledge at design time and do not lend themselves to later extension.

In knowledge representation, ontologies have received widespread attention in particular around the semantic web and for web service management [8, 3], but also innumerable other fields of information science. Ontologies strike a compromise between formalism and expressiveness that allows for being intuitive, generic, extensible, and powerful for reasoning and querying [14]. Zhou and associates apply this approach to the protocol domain [15] for a dynamic protocol framework in which protocols are combined based on their corresponding semantic information. However, this framework centers on a classic stack design of monolithic protocols ignoring decomposition and protocol extensions. Their protocol ontology is based on the Internet protocol layers to reduce redundancy in the description and to limit the complexity of the orchestration process. Consequently, this approach is not fully extensible and cannot support non-classic

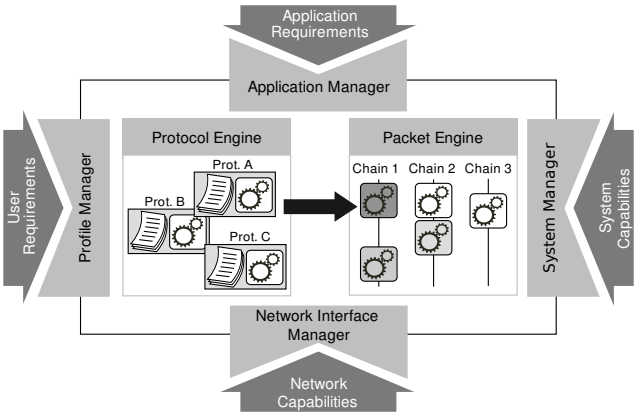

Figure 1: In accordance with external and internal requirements and capabilities, Adapt dynamically orchestrates individual protocol components into protocol chains based on semantic descriptions.

protocol arrangements such as overlay routing, tunneling, layer-crossing routing protocols (common in MANETs), and cross-layer signaling, many of which are particularly desirable for mobile communication. ADAPT lifts this restriction by removing the layer structure from the protocol model. Thus, the complexity of protocol orchestration increases significantly and forms the challenge that we address in this paper.

3. DESIGN

This section briefly covers an overview of ADAPT before detailing the semantic protocol management. It then addresses the criteria influencing protocol orchestration, the semantic protocol model, and the mechanisms of the orchestration process.

As illustrated in Figure 1, the ADAPT end-system architecture is a dynamic protocol framework in the spirit of the x-Kernel or F-CSS. All protocols share a common interface and can - at the software component level - form arbitrary combinations that we call *protocol chains*. The architecture instantiates and configures them dynamically at run time for each communication relationship.

The factors that influence protocol composition are the demands of the application, the user, the capabilities of the network and the capabilities of the whole system. A dedicated manager component monitors each of them and provides a semantic representation for them.

Although such a modularized design of ADAPT encourages protocol decomposition into finer-grained components, we deliberately target complete protocols so as to be able to re-use legacy implementations. Nevertheless, keeping distinct functionality in individual components is essential for combining protocols freely, e.g., to use UDP on top of overlay routing instead of plain IP.

For users to benefit from new or specialized protocols, they need to become available at run time similar to browser plug-ins rather than through a long-term cycle of standardization and operating system releases. Furthermore, such an approach enables other stakeholders like network administrators or ISPs to promote protocols to their users, for example a multicast protocol that reduces an ISP's peering costs by reducing in- or outbound traffic. The distribution of a pro-

tocol as a software component, including digital signatures, and its integration into the protocol manager as a dynamically loadable object can be considered straightforward. To enable the orchestration of arbitrary protocols, each of them is always accompanied by a corresponding document which semantically specifies the functionalities, dependencies, and requirements. Based on this information, ADAPT constructs functionally sound protocol chains from existing and previously unknown protocol components as discussed in the following sections.

3.1 Orchestration Criteria

Two factors drive the process of protocol orchestration in ADAPT: on the one hand, the functionality and interdependencies of the protocols and, on the other hand, the properties of the execution and network environment. These requirements and capabilities fall into the four broad categories outlined in Figure 1. Although many of them relate to typical QoS parameters, our research focus lies on supporting arbitrary protocols at all layers, not on QoS per se.

1. *Network capabilities* primarily influence the orchestration of the network-related lower-level elements of chains. The most immediate influence is exercised by the local network access, such as the available hardware interfaces and their MAC protocols. Remote factors cover aspects such as the routing protocol used in a network or the necessity of authentication or tunneling (e.g., via a VPN) to access the local network or the Internet. To cover QoS aspects during protocol orchestration, ADAPT can gather qualitative information about network characteristics, such as maximum throughput, link loss rates, and latencies.

2. *Device capabilities* are of a similar qualitative nature and primarily reflect information about the available CPU, memory, and energy resources.

3. *Application requirements* are explicitly specified by applications when they request a new session to be established. Typically, they are made up of functional requirements on the new protocol chain, e.g., that it needs to counter packet loss and re-ordering in which case a variant of TCP might eventually become part of this chain. Applications may also include qualitative aspects such as a preference for low latency or more abstract requirements such as privacy. This flexibility stems from the fact that such semantic aspects are fully encoded in the protocol descriptions and do not depend on pre-defined functionality in the ADAPT architecture.

   Traditionally, legacy applications distinguish only between UDP's unreliable datagram transport service and TCP's reliable stream transport service. However, we plan to introduce two interfaces for customized applications: The query interface allows to concisely specify the most common requirements like a transport service type and security and QoS parameters. Through the orchestration interface, applications directly specify protocol orchestration queries with the protocol manager to control any aspect of the orchestration process. Implicit application requirements are inferred by the application manager about applications based on pre-configured knowledge. On mobile devices for example, streaming or terminal applications with long-standing sessions would benefit from instantiating a mobility management protocol such as HIP in their protocol chains.

4. *User preferences* influence how the above qualitative factors are traded against each other in the orchestration process to give users control over ADAPT's orchestration decisions. Typical trade-offs concern security, cost, and performance aspects. User preferences also provide immediate configuration information, for example fixed IP addresses, authentication information, and wireless network priorities. Finally, they allow users to influence the orchestration process such that, e.g., VPN tunneling is enforced for specific networks.

3.2 Orchestration Process

The process of protocol orchestration aims to determine the protocol chain best suited to the given application and environment requirements. Finding such a chain in the full set of possible protocol combinations (a selective approach) suffers from limited scalability with a growing number of protocols. ADAPT thus follows a constructive approach in two stages. It first composes only the protocol chains that are functionally viable and fulfill all requirements imposed by the application request and the current execution environment. In the second stage, an expert systems ranks each resulting chain to determine the one that matches the environment best.

3.2.1 Composition

In a three-step process, the semantic composer relies on different types of information from the protocol ontology to compose valid protocol chains. First, it evaluates the functional requirements of the application and the execution environment to obtain all protocols that are necessary to satisfy these demands. Next, it recursively resolves the dependencies among protocols and constructs partial protocol chains (*stubs*) from this information. Finally, it merges the partial chains into complete functional compounds that can later be instantiated for packet processing.

In the first step, the application sets up a connection (or creates a socket, in legacy terms) and can, in a very simple example, indicate *congestion control*, *datagram delivery*, and *IPv4 addressing* as its requirements to the semantic composer. To bootstrap the composition process, ADAPT treats the application like any other protocol module with its requirements. It distinguishes two classes of requirements: direct and indirect ones. Direct requirements of a protocol P need to be satisfied by the next element Q in a protocol chain and typically relate to interface properties (as for *datagram delivery* and *IPv4 addressing* in this example). Indirect requirements of P impose a looser restriction on the construction process since they need to be honored by an arbitrary element following P in the protocol chain. They usually express more abstract, functional dependencies, such as on *congestion control* in the above example.

The second step is for the semantic composer to create chain stubs by resolving direct requirements. Assume that the protocol modules UDP, DCCP, and Friendly P2P (an application-level congestion control protocol) satisfy the application requirements. In that case, three stubs would be created: App – UDP, App – DCCP, and App – Friendly P2P. The composition process proceeds recursively on the direct

requirements, forking stubs where different alternatives are available to resolve requirements. For each stub, it also accumulates the indirect dependencies of each stub element. The recursion continues on each stub until a link-layer protocol module is added which completes the chain because it effectively connects the chain to a network interface. If a stub cannot be completed in this manner because no available protocol satisfies the given direct dependencies, the stub is discarded immediately. Additionally, the semantic composer performs loop detection to eliminate degenerate stubs consisting, e.g., of recursively nested IP GRE tunnels[1].

The third step in the composition process is to evaluate the indirect requirements. If the (lower-layer) protocols in a completed chain satisfy the accumulated indirect requirements of the other (higher-layer) protocols, the chain is valid and is passed on to the ranking phase in the orchestration process. This applies, for example, to the chain App – DCCP – IPv4 – Eth, but not to App – UDP – IPv4 – Eth since it does not meet the *congestion control* requirement. The latter chain is thus discarded.

Changes in the execution or network environment of a device can cause existing protocols chains to be re-evaluated and re-composed if necessary. In such a case, the protocol manager first derives from the protocol descriptions, which chain modifications are valid at run time. For example, a standard established TCP / IP / Ethernet connection cannot both maintain its TCP state and switch to a different IP instance with a different IP address. However, a switch of the existing TCP / IP instances to a new Ethernet instance, e.g., on a wireless device, is feasible. This information about re-configurability is represented as additional protocol dependencies and fed as such to the semantic composer.

Exchanging protocol instances during run time leads to the problem of whether and how to convert and transfer state information between these instances. In this paper, we do not address this issue but focus on how to select appropriate protocol combinations that do not depend on such state transitions.

3.2.2 Ranking

In the second stage of protocol orchestration, an expert system ranks the constructed chains according to quality metrics and user preferences to determine a single chain to instantiate. The network interface and device managers provide the base information about the current execution and network environment. The equivalent information about protocols is derived from their individual descriptions (e.g., the loss rate range acceptable to or the computational overhead imposed by a protocol). Additionally, the application and profile managers may provide weight factors for each of these properties to bias their influence on the result. Based on these inputs, the expert system matches the corresponding protocol and environment properties to derive specific metrics. Then, it feeds them to a multi-criterion decision system that calculates a ranking order from the metrics and their weight factors. Thus, the highest-ranking protocol chain emerges as the best match for the current communication requirements.

3.2.3 Instantiation

[1]If necessary, such constructs can however explicitly requested before stub creation.

The overall orchestration process is triggered when an application requests a new connection to be created. The protocol manager first checks whether the application request or the profile manager demand a specific chain layout. If so, it instantiates this preset, otherwise the semantically driven composition continues as described above. The protocol manager passes the resulting chain in a simple descriptive format to the packet manager which creates a session instance of each protocol. The packet manager links the instances to each other and associates the resulting chain with the application that triggered the connection setup. At this point, the protocol chain is fully established and available to its application. An instantiated chain is the equivalent of a socket instance in traditional network stacks. In contrast to those, chains contain only session information and activation hooks for their associated protocols and they allow to exchange individual protocol sessions at run time to support dynamic reconfiguration. If the protocol manager is not able to compose any valid protocol chain for the given requirements, this fact is signalled to the application and the user so they can relax their original requirements.

Incoming network packets carry identifiers that determine which protocols need to process the packets. This applies both to today's layered protocols and to approaches in the spirit of the Role-Based Architecture. To map a packet to its associated session, ADAPT first derives for each protocol identifier in the packet a protocol-specific de-multiplexer (e.g., that of TCP), which in turn provides the matching session instance (e.g., a TCP session as derived from the port number). Similarly, session initiating packets trigger the protocol-specific de-multiplexer to create a new session instance on the fly.

3.3 Semantic Protocol Modeling

ADAPT models protocols and the execution environment in a class-oriented ontology. The ontology allows to represent, share, and extend knowledge about the domain of communication protocols. It contains the formal descriptions of each protocol model and provides the basis for automated reasoning.

Similar to object-oriented software design, ontologies typically structure knowledge into hierarchies of classes with each class having distinct properties. As depicted in Figure 2, the protocol ontology models all elements as classes including the relationships between protocols. Although such information can be represented by class properties, the introduction of additional knowledge might then require modifying existing classes. Since that would hamper the extensibility of the models, we aim to model all information as classes. Figure 2 illustrates this, for example, for TCP's indirect dependency on eventual medium access and its direct dependency on an IPv4-style routing-level address. The indirection through the tcpDep1 class allows to add a second set of dependencies on medium access and an IPv6-style address (not depicted) to the ontology without modifying the TCP class itself.

To support the orchestration process and its individual stages, the ontology distinguishes between three distinct categories: functionalities, dependencies, and qualitative information. The functionality model derives directly from individual protocol functionalities, such as session support, packet loss handling, or local or global routing. By subclassing, it expresses a refinement of a more abstract func-

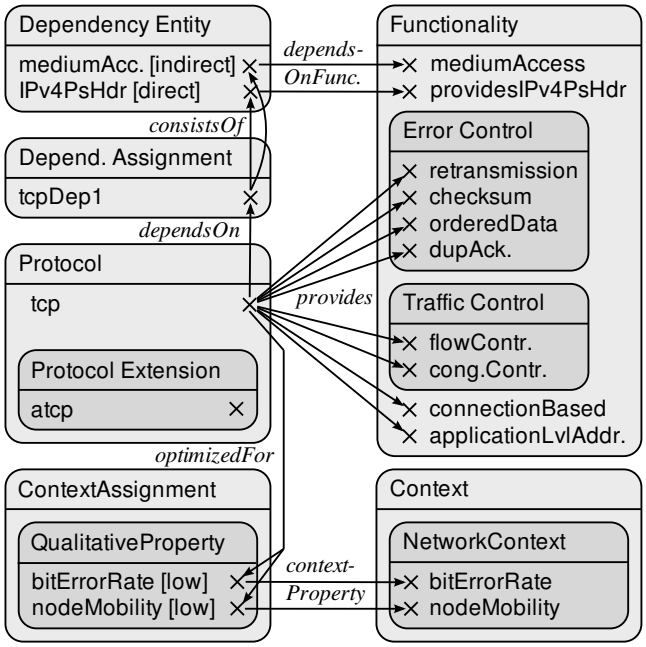

Figure 2: Adapt's semantic model of TCP (excerpt). Each item in a box represents a class and arrows represent relationships between classes.

tionality (e.g., packet retransmission could be a sub-class of the loss handling class). The dependency model establishes associations between protocols, functionalities, and user-defined criteria It can also establish *and* and *or* relationships between multiple dependencies (e.g., to enforce the inclusion of one of two specific encryption protocols). Furthermore, the ontology provides *qualitative information* about such aspects as protocol resource demands. Here, we distinguish between *requirements* a protocol imposes on its protocol chain or the environment (e.g., the existence of a DNS server) and information that solely affects the *ranking* of different protocol chains.

ADAPT uses *OWL DL*, a sublanguage of the *OWL Web Ontology Language* [4], that is based on description logic and thus provides high expressiveness while still maintaining completeness and decidability for reasoning systems. In OWL DL, knowledge is represented by classes, individuals belonging to classes, and relations between them. Classes in an ontology form a hierarchy, which is either asserted manually or can be automatically inferred for classes subsuming others. Similarly, individuals form instances of classes either explicitly through manual assertion or implicitly through inference based on restrictions imposed by logic expressions.

Based on the explicitly defined *asserted model*, the reasoning process derives an *inferred model* that represents additional knowledge. ADAPT employs the following reasoning capabilities, primarily as fundamental means to integrate future protocols, orchestration criteria, and metrics:

Type inheritance: by inference, individuals of class X inherit the types of X's super classes. Thus, protocols describe their functionality precisely (e.g., RSA encryption) and can later be classified more generically (e.g., as providing confidentiality) through newly introduced knowledge.

Inverse properties: a symmetric relation between X and Y specified only for X is inferred to apply to Y. Thus, the inferred knowledge base remains consistent despite the incorporation of new knowledge.

Instance classification: *defined classes* classify individuals through a set of logic expressions. Consequently, the knowledge about the criteria of class membership receives an explicit representation.

Rule support: user-defined rules allow to assert new facts about individuals. Protocols can be asserted to support reliability, for example, if they provide ordered data delivery, retransmission, and a checksum algorithm.

OWL DL represents information either in an abstract syntax or *RDF/XML* format. It allows to describe protocol semantics as a single unit which can be easily merged with a pre-existing ontology. ADAPT relies on this feature to integrate both the functionality and the semantics of new protocols on end systems at run time.

Query	Matching	Compos.	Sum	# chains
Reliability	43 ms	117 ms	166 ms	6
Name res.	30 ms	136 ms	166 ms	4
Multicast	53 ms	62 ms	115 ms	18

Table 1: Protocol composition based on restrictive queries for reliability support, name resolution support, and multicast support

Query	Matching	Compos.	Sum	# chains
Reliability	33 ms	222 ms	255 ms	226
Name res.	29 ms	375 ms	404 ms	655
Multicast	54 ms	112 ms	166 ms	88

Table 2: Protocol composition based on non-restrictive queries

4. IMPLEMENTATION

The implementation of ADAPT aims at a system that can be used with standard applications in realistic mobile communication scenarios. Due to the popularity of Windows-based mobile devices, the main target platform of ADAPT are the desktop and mobile version of Windows. The protocol framework is realized as a user-level application that transparently intercepts network packets and socket operations of legacy applications via small driver components in the operating system. Furthermore, we supplant the operating system's TCP/IP stack with protocol modules for TCP, UDP, and IP based on existing user-level implementations with further protocols like SCTP, IPSec, overlay and VPN protocols to follow.

The protocol manager contains a matchmaker, a composition engine, and a stub expert system. It is implemented in Java to access the description repository with the Jena semantic web framework which manages the ontology and provides a reasoner, a query engine, and basic rule support. We use SPARQL [11] to perform queries on the description repository. SPARQL is an SQL-like query language based on graph patterns which efficiently operates on RDF triplets without being aware of OWL semantics. Hence, we designed the ontology for a two-stage execution model: At first, a DL reasoner infers knowledge based on the full OWL DL semantics. This is a complex operation but is only necessary when initializing or extending the ontology, e.g., with a new protocol. Subsequently, the semantic composer uses efficient SPARQL queries on the inferred model during the orches-

tration process. Performance can be further improved by caching the stub chains from previously resolved dependencies as this reduces the number of necessary queries during protocol composition.

As an initial evaluation, we tested the semantic composer with three typical functionality requests: Support for a reliable connection, for multicast, and for name resolution. The ontological model contained 14 protocols as well as protocol extensions (e.g., an extension to TCP that makes it perform better in scenarios with high bit error rate), tunnels (e.g., in case IP Multicast is not supported by the network the host resides in), and network requirements (existence of a DNS server). All measurements were performed on a Linux system with a 1.80 GHz Intel Pentium M processor and 1GB RAM, employing Sun's Java 1.5.0 run-time environment and version 2.5.5 of the Jena library.

Table 1 lists the duration of the matchmaking process, the protocol composition process, and the number of resulting protocol chains for queries that specify the desired functionality concisely. More loosely specified queries are satisfied by more chains but they also increase the composition time significantly, as Table 2 illustrates. We see considerable room for improvement in our current implementation, e.g., via additional, though less generic, rules in the composition process, pre-computation, and caching of chain stubs. Although an evaluation of the expert system is still outstanding, these initial results show the practical viability of a functional composition of protocol modules based on ontological models.

5. CONCLUSION

The benefits of dynamic protocol frameworks are well-known and particularly attractive in mobile communication scenarios. Given such a framework, the orchestration of individual protocols into a compound that is functionally and qualitatively sound becomes a non-trivial task because it depends on the semantics of the components and their environment. In this paper, we make a case for modeling these semantics through ontologies and show how the composition process can leverage such an ontology. Our results suggest that this approach can form a viable basis for automatically orchestrating communication stacks based on the functionality provided by individual components instead of rigid design criteria like network layers.

6. REFERENCES

[1] R. Braden, T. Faber, and M. Handley. From Protocol Stack to Protocol Heap: Role-based Architecture. *SIGCOMM Comput. Commun. Rev.*, 33(1), 2003.

[2] N. C. Hutchinson and L. L. Peterson. The x-Kernel: An Architecture for Implementing Network Protocols. *IEEE Transactions on Software Engineering*, 17(1), 1991.

[3] L. Li and I. Horrocks. A Software Framework for Matchmaking Based on Semantic Web Technology. In *Proceedings of the 12th International Conference on World Wide Web*, 2003.

[4] D. L. McGuinness and F. van Harmelen. OWL Web Ontology Language Overview. W3C Recommendation, 2004.

[5] R. Moskowitz, P. Nikander, P. Jokela, and T. Henderson. Host Identity Protocol. RFC 5201 (Experimental), 2008.

[6] M. Muhugusa, G. Di Marzo, C. F. Tschudin, and J. Harms. ComScript: An Environment for the Implementation of Protocol Stacks and their Dynamic Reconfiguration. In *International Symposium on Applied Corporate Computing ISACC 94*, 1994.

[7] S. W. O'Malley and L. L. Peterson. A Dynamic Network Architecture. *ACM Transactions on Computer Systems*, 10(2), 1992.

[8] M. Paolucci, T. Kawamura, T. R. Payne, and K. P. Sycara. Semantic Matching of Web Services Capabilities. In *ISWC '02: Proceedings of the First International Semantic Web Conference on The Semantic Web*, 2002.

[9] L. Peterson, N. Hutchinson, S. O'Malley, and M. Abbott. RPC in the x-Kernel: Evaluating New Design Techniques. In *Proceedings of the Twelfth ACM Symposium on Operating Systems Principles*, 1989.

[10] T. Plagemann, M. Vogt, B. Plattner, and T. Walter. Modules as Building Blocks for Protocol Configuration. In *Proceedings of the International Conference on Network Protocols*, 1993.

[11] E. Prud'hommeaux and A. Seaborne. SPARQL Query Language for RDF. W3C Recommendation, 2008.

[12] R. Stewart. Stream Control Transmission Protocol. RFC 4960 (Proposed Standard), 2007.

[13] C. Tschudin. Flexible Protocol Stacks. In *SIGCOMM '91: Proceedings of the conference on Communications architecture & protocols*, 1991.

[14] X. H. Wang, D. Q. Zhang, T. Gu, and H. K. Pung. Ontology Based Context Modeling and Reasoning Using OWL. In *Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops*, 2004.

[15] L. Zhou, H. K. Pung, L. H. Ngoh, and T. Gu. Ontology Modeling of a Dynamic Protocol Stack. In *31st IEEE Conference on Local Computer Networks*, 2006.

[16] M. Zitterbart, B. Stiller, and A. N. Tantawy. A Model for Flexible High-performance Communication Subsystems. *Selected Areas in Communications, IEEE Journal on*, 11(4), 1993.